Claims

I claim:

1. (Currently amended) A method for creating a linked list data structure to which prefetching can be applied in order to minimize the number cache misses endured during traversal, said method comprising the steps of:

creating a parallel data structure consisting of a plurality of partitions (N) consisting of a plurality of sublists (P), associating a state vector (S) with the data structure to maintain the state of the traversal of each sublist, and maintaining the state of the last sublist to which an element is added in a variable (H), whereby; additions are made to the

the head of the list by decreasing the list head index to H – 1 modulo P and adding new nodes to the head of the list indexed by the thus updated value of the head index.

pipelining the traversal across the *N* partitions of the data structure;

determining the prefetch distance required in order to traverse said data structure using the aforementioned pipelined traversal, said prefetch distance being determined ~~experimentally~~ by the programmer, computed using prior art, or by the compiler;

inserting prefetch instructions into the traversal loop body.

2. (Original) The method for creating a data structure according to claim 1 wherein the data structure is provided as part of a library.

3. (Original) The method for creating a data structure according to claim 1 wherein a data structure defined by an application is transformed by a compiler, said compiler performing the necessary steps to modify corresponding traversal code to perform software pipelined prefetching.

4. (Original) The method of creating a data structure according to claim 1 wherein the data structure is generated by a set of macros.

5. (Original) The method of creating a data structure according to claim 1 wherein the method of creating such a data structure is supplied as part of a class library in an object oriented system.

6. (Original) The method of creating a data structure according to claim 1 wherein the steady state loop is preceded by a prologue which performs no data structure traversal, but which does generate prefetch instructions.

7. (Original) The method of creating a data structure according to claim 1 wherein the steady state loop is followed by an epilogue in which no prefetch instructions are performed, but in which traversal of the data structure continues and possibly completes.

8. (Canceled)

~~7. The method of constructing a linked list according to claim 1 wherein said linked list is partitioned into a plurality of sublists (P), a state vector (S) is associated with the list to maintain the state of the traversal of each sublist, and the state of the last sublist to which an element is added is maintained as a variable (H).~~

9. (Currently amended) The method of constructing a linked list according to claim 8-1 wherein additions to the end of the list are added to the end of the sublist indexed by I modulo P and the list header index H updated to H + 1 modulo P.

10. (Canceled)

~~9. The method of constructing a linked list according to claim 8 wherein additions to the head of the list are added by decreasing the list head index to H – 1 modulo P and adding new nodes to the head of the list indexed by the thus updated value of the head index.~~

11. (Currently amended) The method of constructing a linked list according to claim 8-1 wherein
    a variable (T) holds the index of the last sublist to have an element inserted,
    an array containing P pointers, said pointers linking to the tail of each sublist,
    insertion at the tail of the list is performed by the method consisting of
        updating the list tail index (T) to T + 1 modulo P
        and adding a node to the sublist indicated by the new value of T,
    and whereby deletion from the tail of the sublist is performed by the method consisting of
        updating the list tail index (T) to T – 1 modulo P
        and deleting the node from the tail of the sublist indicated by the new value of T.

12. (Currently amended) The method of traversing the linked list constructed according to claim 8-1 whereby the linked list is traversed by the method comprising the steps of
        prefetching the head of each sublist;
        initializing a vector (S) with the head of each sublist;
        traversing the list in a software pipelined manner by
            iterating over S and processing each node held in S,
            performing the requested work on each node held in S,
            updating each element of S to hold the next element in the corresponding sublist,
            and issuing a prefetch for the next node in each sublist.

13. (Original) The method of traversing a tree according to the method of claim 1 wherein a tree is constructed as a forest of trees.

14. (Original) The method of constructing a pre-order traversal tree according to the method of claim 1 comprising
    a plurality of trees (P);
    an array in which to store a pointer to the root node of each tree;
    a variable (T) holding the value of the last tree into which a node was inserted;
    the method of adding a node to the forest by assigning the value T + 1 modulo P to the variable T and using a normal insertion into the tree indexed by the value of T thus updated;
    the method of traversing the forest in pre-order fashion.

15. (Currently amended) The method of traversing a single tree by creating a forest of subtrees by the method comprising the steps of:

initiating a level-order traversal starting at the root,

maintaining an array of pointers to nodes in the tree in the course of the level-order traversal,

discontinuing the level-order traversal when a number of subtrees sufficient for effective software pipelined traversal has been achieved, the aforementioned array of pointers thereby containing the pointers to the roots of the trees of a forest to which software pipelined traversal can be applied,

then proceeding with a traversal, ~~according to claim 13,~~ wherein a tree is constructed as a forest of trees, ~~wherein~~ and subtrees pointed to by the aforementioned array of subtrees constitute the forest across which software pipelined traversals are performed.

16. (Original) The method of traversing a single tree according to claim ~~15~~15 wherein the children of each node encountered in the course of the level order traversal are prefetched.

17. (Original) The method of deconstructing a tree into a forest ~~according to claim~~ according to claim ~~15~~ whereby

a queue is used to hold the nodes of the current level of an in-order traversal of the tree, said nodes being added to said queue as they are encountered in the course of the traversal of each level,

a second queue is used to hold pointers to the children of the nodes at the current level,

said queues swap roles as the traversal transitions from one level to the next,

and the forest is constructed from the subtrees rooted at the nodes pointed to by the contents of the aforementioned two queues.

18. (Original) The method of traversing a single tree according to claim ~~17~~17 wherein the children of each node encountered in the course of the level order traversal are prefetched.